Learning Relational Dynamics of Stochastic Domains for Planning

David Martínez and **Guillem Alenyà** and **Carme Torras**
Institut de Robòtica i Informàtica Industrial (CSIC-UPC)
{dmartinez,galenya,torras}@iri.upc.edu

Tony Ribeiro
IRCCyN, École Centrale de Nantes
tony.ribeiro@irccyn.ec-nantes.fr

Katsumi Inoue
National Institute of Informatics, Japan
inoue@nii.ac.jp

Abstract

Probabilistic planners are very flexible tools that can provide good solutions for difficult tasks. However, they rely on a model of the domain, which may be costly to either hand code or automatically learn for complex tasks. We propose a new learning approach that (a) requires only a set of state transitions to learn the model; (b) can cope with uncertainty in the effects; (c) uses a relational representation to generalize over different objects; and (d) in addition to action effects, it can also learn exogenous effects that are not related to any action, e.g., moving objects, endogenous growth and natural development. The proposed learning approach combines a multi-valued variant of inductive logic programming for the generation of candidate models, with an optimization method to select the best set of planning operators to model a problem. Finally, experimental validation is provided that shows improvements over previous work.

1 Introduction

Recent planners are able to solve complex probabilistic tasks given a state representation and a model (Kolobov et al. 2012; Keller and Eyerich 2012), and they have been applied successfully in several fields such as robotics (Kulick et al. 2013; Martínez, Alenyà, and Torras 2015) and scheduling (Zhu, Lizotte, and Hoey 2014). However, these planners rely on a model. In this paper we propose a new method to learn a probabilistic relational model including action and exogenous effects from a set of completely observable state transitions. We consider that action effects are those that occur when the agent executes an action (e.g. a robot moves to a new position, the robot grabs an object), while exogenous effects are those that do not depend on an action (e.g. people moving in the street, a traffic light turns red, a plant grows).

There exist previous works that tackle the problem of learning models. Some of them estimate the parameters of a model (Thon, Landwehr, and De Raedt 2011; Thon et al. 2009), but the model has to be provided and only its parameters are learned. A more complete approach (Sykes et al. 2013) learns probabilistic logic programs, but the restrictions for the initial set of candidate rules need to be manually coded. In contrast, we learn the complete model and no restrictions are required.

Most approaches that learn complete models handle deterministic tasks, and although they can tackle partial observability (Mourao et al. 2012; Zhuo and Kambhampati 2013) or apply transfer learning (Zhuo and Yang 2014), they do not consider uncertain effects. In this work we focus on models with uncertain effects.The most similar approaches to ours are those that learn relational action models with uncertain effects (Pasula, Zettlemoyer, and Kaelbling 2007; Deshpande et al. 2007; Mourao 2014). They learn the effects that each action may have for each set of preconditions, which are then represented with probabilistic STRIPS-like models. However, they do not learn exogenous effects that are not related to any action. These methods cannot be easily extended to learn exogenous effects. Pasula et al., and Deshpande et al. use a local search algorithm that works well when transitions are explained by one rule, but faces many local minima when tackling domains with exogenous effects, as two or more new rules have to be added to properly explain a transition. Mourao et al.'s approach exhibits a similar problem since it learns one rule per transition. We take a different approach that learns models with uncertain effects, and also those with exogenous effects.

The problem of learning minimal effects from a log of input data transitions is known to be NP-Hard (Walsh 2010). The approaches shown before, as well as our approach, apply heuristics to find solutions with any number of input experiences. Optimal approaches that learn complete probabilistic models have also been proposed (Walsh et al. 2009), but they require too many input experiences.

To summarize, we propose a novel method that takes as input a set of state transitions, and learns a relational model with probabilistic and exogenous effects to be used for planning. The learned model will consist of a set of planning operators that define the different effects. The proposed method can be divided into three parts:

- *Candidate planning operator generation*. Candidates are generated with the LFIT (Learning From Interpretation Transitions) framework (Inoue, Ribeiro, and Sakama 2014). LFIT induces a set of propositional rules that realize the given input transitions. Specifically, an algorithm that guarantees to learn the set of minimal rules is used (Ribeiro and Inoue 2014).
- *Planning operator selection*. To select the best subset of

candidates, we define a score function that is maximized by candidates that explain input state transitions while being general enough. Based on this score function, a search optimization method guided by an heuristic function is proposed. Moreover, suboptimal solutions to make complex tasks tractable are provided.

- *Data transformations*. Our approach combines (a) LFIT on the propositional level to ensure that candidates are minimal, (b) an optimization method that works on the relational level to apply relational generalizations when selecting subsets, and (c) grounded input data. Since, as mentioned, the approach requires three different types of data (grounded, relational and propositional), data transformation methods are needed.

2 Background

Two types of representations are combined in this work, a relational one for the planning operators, and a propositional one for the propositional rules. We assume complete observability and uncertain effects.

Relational Formulation

Literals l_i are expressions of the form $(\neg)p(t_1, ..., t_m)$ where p is a predicate symbol, $(\neg)$ represents that the atom may be optionally negated, and t_i are the terms. Terms can be variables, which have a preceding "?" symbol (e.g. ?X), and objects, which are represented without an "?" symbol (e.g. box1). We use a relational representation where expressions take objects as arguments to define their grounded counterparts. A state s is defined as a conjunction of grounded literals that follow the closed world assumption $s = l_1^g, ..., l_N^g$.

A planning operator $o \in \mathcal{O}$ defines how a literal changes based on a set of preconditions. Operators take the form

$$o(t_1, ..., t_n) = l_h : p_o \leftarrow l_1 \wedge \cdots \wedge l_m, (a) \qquad (1)$$

where l_h is the head of the operator, p_o is the probability of l_h being in the next state given that the body and the action are satisfied, $l_1 \wedge \cdots \wedge l_m$ are the literals in the body, (a) is an optional action, and t_i are the terms that may appear in the head, body and action. The action is optional so that operators can capture both action effects when there is an action and exogenous effects when there is no action. Note that operators are not Horn clauses as negation can appear in both the body and the head.

A grounded operator only has objects as terms. If an operator o has n variables, its groundings $Gr(o)$ are a set of operators, each taking one of the possible combinations of n objects.

Example 1 *Having the objects $\{a, b, c\}$ and the operator $o1(?X, ?Y) = at(?Y) : 0.8 \leftarrow road(?X, ?Y) \wedge at(?X)$, the possible groundings can be obtained by substituting $?X$ and $?Y$ for every permutation of 2 objects. One grounding would be: $o1(a, c) = at(c) : 0.8 \leftarrow road(a, c) \wedge at(a)$.*

The transition dynamics are defined by a set of planning operators $\mathcal{O}$. A grounded operator o_g is said to cover a state-action pair (s, a) when the literals of the body are in s, and the optional action of the operator is either a, or the operator has no action: $cov(o_g, s, a) = (body(o_g) \subset s) \wedge ((action(o_g) = a) \vee (action(o_g) = \varnothing))$.

Likewise, a non-grounded operator o covers (s, a) if $cov(o, s, a) = \exists\, o_g \in Gr(o) \mid cov(o_g, s, a)$ holds.

A successor state s' is obtained by applying all groundings of all operators to (s, a, s'). When a grounded operator o_g is applied to (s, a, s'), its head is added to the state s' with a probability p_{o_g} if $cov(o_g, s, a)$. We require operators to be mutually exclusive, there cannot be two operators with the same head atom that cover the same (s, a) as their heads may conflict. One example of such conflict would be $[o_{g,1}(r1) = at(r1) : 0.8 \leftarrow ...]$ and $[o_{g,2}(r1) = \neg at(r1) : 0.6 \leftarrow ...]$ where both heads cannot hold at the same time as one contradicts the other. The objective of this work is to learn models that can be used by planners, and planners require conflict-free operators. A planner has to know precisely the expected effects of applying a planning operator. If two different operators were to make conflicting changes, the effects would be undefined.

If s' is a successor state of s, we define $changes(s, s')$ as the set of literals $\{c \in s', c \notin s\}$. Given a state-action pair (s, a) and a grounded operator o_g, a transition change $c \in changes(s, s')$ has a likelihood

$$P(c \mid o_g) = \begin{cases} p_{o_g}, & cov(o_g, s, a) \wedge (c = head(o_g)) \\ 0, & \text{otherwise.} \end{cases} \qquad (2)$$

And a set of non-grounded operators $\mathcal{O}$ gives the following likelihood to a change c:

$$P(c|\mathcal{O}) = \begin{cases} P(c|o_g), & \exists!\, o_g \in Gr(\mathcal{O}) \mid P(c|o_g) > 0 \\ 0, & \text{otherwise,} \end{cases} \qquad (3)$$

where $\exists!$ is the operator for uniqueness quantification. If more than one operator covers the same change given the same state-action pair, there is a conflict and the behavior is undefined, so a likelihood of 0 is given.

Propositional Formulation

On the propositional level, multi-valued atoms m_i are expressions of the form $p = x$, where x is the value of the atom, and the atoms have no terms. A state is a conjunction of propositional multi-valued atoms $s = m_1, m_2, ..., m_n$ where every predicate must appear only once (i.e. $\forall\, m_i \in s,\ \nexists\, x, y \mid ((m_i = x) \wedge (m_i = y)),\ x \neq y$). LFIT learns a set of propositional rules that take the form

$$r = m_h : p_r \ \leftarrow\ m_1 \wedge \cdots \wedge m_n \qquad (4)$$

where m_h is the head of the rule, p_r the probability, and $m_1 \wedge \cdots \wedge m_n$ is the body of the rule.

3 Method

In this section we show how the planning operators are learned. The input is a set T of training transitions which are triples $t = (s, a, s')$ where s' is a successor state of s when the action a was executed. The output is a set of planning operators $\mathcal{O}$ that define the model. Figure 1 shows the different steps of the algorithm, which are described below:

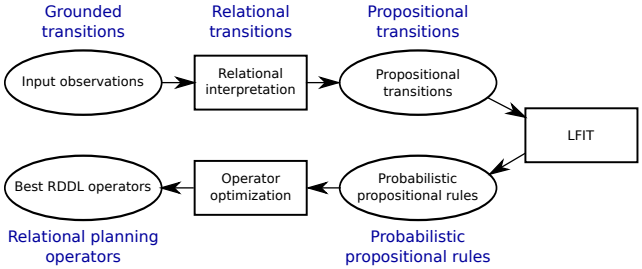

Figure 1: Data representation used for each module. The input and output data are shown in ellipses, and the processing modules are shown in boxes. The data representation used is indicated at each step.

- Transform grounded transitions to relational (non-grounded) ones. The objective is to learn relational operators that can take objects as arguments to generalize. Note that grounded literals are different from propositional ones because the objects and the variables in their terms can be identified. The transformation to relational transitions requires to substitute objects with variables.
- Obtain candidate operators. LFIT is used to obtain all possible candidate operators for a given set of transitions. The main advantage of using LFIT is that it obtains the set of minimal rules that model both action and exogenous effects. To use LFIT, first the relational transitions have to be transformed to propositional ones, and later, the output propositional rules to planning operators.
- Select the subset of candidate planning operators that best models the training transitions. This is detailed in Sec. 4.

The aim of these transformations is to generalize better. The grounded input transitions could be straightforwardly transformed to propositional ones, but LFIT would learn rules that would model grounded data with no relational generalizations. When propositional transitions are obtained from relational ones, LFIT learns rules that model relational data. A relational representation is more compact and general as an infinite number of objects can be represented by each variable. The trade-off of learning relational generalizations is that the number of generated relational transitions is larger than the number of input grounded transitions, which increases the learning time.

Grounded to Relational Transitions

The goal is to obtain a relational representation that can generalize to different objects. If the dynamics of an object are learned, the same dynamics can be applied to other objects, not requiring examples of every possible grounding.

Since generating all possible relational combinations of every transition would be highly inefficient, we limit the number of relational variables to a fixed number ω, which imposes a limit on the maximum number of variables that learned operators will have. Selecting the right value of ω is important. To learn effects that involve n objects, a value $\omega \geq n$ is required. However, the number of relational transitions scales exponentially with ω, thus a large value of ω is intractable.

This module generates all possible relational transitions with at most ω variables. For every transition $t \in T$, the following method is applied:

- *Input*: grounded transition $t = (s, a, s')$, max variables ω. We define the objects in s as $b_{s,i}$ and the objects in a as $b_{a,i}$. The action a has m objects.
- Obtain combinations of ω objects. For each combination of $\omega - m$ objects $(b_{s,1}, \dots b_{s,\omega-m})$ that are not in the action $(b_{s,i} \neq b_{a,j}, \forall\, i, j)$ do:
  - Create $v_{obj} = (b_{a,1}, \dots b_{a,m}, b_{s,1}, \dots b_{s,\omega-m})$ where $(b_{a,1}, \dots b_{a,m})$ are the objects in the action a.
  - Add v_{obj} to V_{obj}.
- For each $v_{obj} \in V_{obj}$:
  - A new transition $t' = t$ is created.
  - Replace in t' all objects in v_{obj} for variables.
  - Remove from t' any remaining literal with objects.
  - Add the new transition t' to T'.
- *Output*: a set of relational transitions T'.

Example 2 *Given a grounded transition* $(s, a \rightarrow s')$ *:*

$$at(r1) \wedge road(r2, r3) \wedge road(r1, r3), move(r3) \rightarrow road(r1, r3) \wedge road(r2, r3) \wedge at(r3),$$

the following relational transitions are generated (ω=2)*:*

$$v_{obj} = (r3, r1): \; at(?Y) \wedge road(?Y, ?X), move(?X) \rightarrow road(?Y, ?X) \wedge at(?X);$$
$$v_{obj} = (r3, r2): \; road(?Y, ?X), move(?X) \rightarrow road(?Y, ?X) \wedge at(?X).$$

Relational to Propositional Transitions

To create the input that LFIT requires, which are pairs (s, s') of propositional states, a library L_{conv} that converts between relational and propositional atoms is created. For each relational atom d_r, a new propositional atom d_p is created and the pair (d_r, d_p) is added to L_{conv}. Using L_{conv}, everything is substituted by its propositional counterpart:

- Relational literals are represented with propositional atoms that take the values 1 (true) or 0 (false).
- Relational transitions are triples (s, a, s'), while propositional transitions are pairs (s, s'). Therefore, an additional multi-valued atom is added to propositional transitions to represent the action. This atom takes as value the corresponding action name in L_{conv}, or "*noaction*" if there is no action. Note that the multi-valued representation is currently only used to model this action atom: other literals are binary, but there may be several different actions (and only one action per transition).

Example 3 *Using the relational transitions in example 2, the following library is created* $L_{conv} = \{(at(?Y), b1), (at(?X), b2), (road(?Y, ?X), b3), (move(?X), b4)\}$*. The propositional transitions obtained by using* L_{conv} *are:*

$$(b1{=}1) \wedge (b3{=}1) \wedge (action{=}b4) \rightarrow (b3{=}1) \wedge (b2{=}1);$$
$$(b3{=}1) \wedge (action{=}b4) \rightarrow (b3{=}1) \wedge (b2{=}1).$$

LFIT

The LFIT framework (Inoue, Ribeiro, and Sakama 2014) is used to obtain the set of probabilistic candidate rules that model the dynamics. Given a batch of propositional transitions (s, s'), LFIT induces a *normal logic program* that realizes the given transitions. This framework has been extended (Ribeiro and Inoue 2014) with a new algorithm that guarantees that the learned rules are minimal: the body of each rule constitutes a prime implicant to infer the head. It is based on a top-down method that generates hypotheses by *specialization* from the most general rules. Moreover, in (Martínez et al. 2015) the framework was adapted to capture also non-deterministic dynamics. Our approach uses the specialization and non-deterministic algorithm of LFIT, so it learns the set of minimal probabilistic rules that models all effects appearing in the input transitions. It learns both action and exogenous effects because the action is just another atom that may or may not appear in the body of a rule.

Propositional Rules to Planning Operators

Planning operators (eq. 1) can be reconstructed from probabilistic rules (eq. 4) by using the library L_{conv} created before. For each propositional rule:

- The atoms in the body and head of the rule are transformed to relational ones (using L_{conv}), and added to the body and head of a planning operator.
- The action is extracted from the multi-valued action atom in the rule body. If the atom value is not "*noaction*", the corresponding action in L_{conv} is added to the operator.

Example 4 *Given that LFIT had learned the following rule*

$$(b2{=}1) : 0.8 \leftarrow (b1{=}1) \wedge (b3{=}1) \wedge (action{=}b4),$$

using the library L_{conv} generated in example 3, the resulting operator $o(?X, ?Y)$ is

$$at(?X) : 0.8 \leftarrow at(?Y) \wedge road(?Y, ?X), move(?X).$$

Traditionally, PPDDL (Younes and Littman 2004) has been the standard language to model probabilistic domains, but it is difficult to model exogenous effects with it. Therefore, our approach uses the RDDL language (Sanner 2010), which has been the standard for the latest probabilistic planning competitions (IPPC 2011 and 2014). Writing our planning operators with RDDL is straightforward, and this language can be used directly by state-of-the art planners. RDDL objects and variables have types, and a variable can only be substituted by an object of the same type. However, for clarity and simplicity, we assume through the paper that there are no types, as adding them is trivial.

4 Planning Operator Selection

LFIT provides the set of minimal rules (that have been transformed to planning operators) that describe all the transitions. Note that LFIT learns the set of minimal rules, and not the minimal set of rules, so many operators may model the same changes and underfit or overfit. The subset of planning operators to model the transition dynamics is selected as follows:

- A score function is defined to evaluate the operators.
- A heuristic search algorithm selects the set of operators that maximizes the score. Note that this set may differ from the actual model, as it depends on the coverage of the input transitions and the quality of the score function.
- The subsumption tree is used to improve efficiency by partitioning the set of candidates into smaller subsets.

Score Function

The score function values the quality of a set of rules. The following functions are used by the score function:

- The likelihood is the probability that a transition $t = (s, a, s')$ is covered by a set of planning operators $\mathcal{O}$:

$$P(t \mid \mathcal{O}) = \prod_{c \in \text{changes}(t)} P(c \mid \mathcal{O}, s, a). \tag{5}$$

- The penalty term $Pen(\mathcal{O}) = \sum_{o \in \mathcal{O}} |body(o)|$ is the number of atoms in the operator bodies.
- The confidence $Conf(T, \epsilon)$ is obtained from Hoeffding's inequality. The probability that an estimate $\widehat{o_{prob}}$ is accurate enough $|\widehat{o_{prob}} - o_{prob}| \leq \epsilon$ with a number of samples $|T|$ is bounded by $Conf(T, \epsilon) \leq 1 - e^{-2\epsilon^2|T|}$.

Finally, the proposed score function is defined as

$$s(\mathcal{O}, T) = \underset{t \in T}{E}[\log(P(t|\mathcal{O}))] - \alpha \frac{Pen(\mathcal{O})}{Conf(T, \epsilon)}, \tag{6}$$

where $\alpha > 0$ is a scaling parameter for the penalty term. This score function is based on Pasula et al.'s one (Pasula, Zettlemoyer, and Kaelbling 2007), where the likelihood is maximized to obtain operators that explain the transitions well, and the penalty term is minimized to prefer general operators when specific ones have very limited contributions. In contrast to Pasula et al.'s approach, the confidence term is added so that the penalty is increased when few transitions are available, as the estimates are less reliable.

Heuristic Search

Given a set of operators $\mathcal{O}$ with the same head, a heuristic search method is used to find the best subset of operators that maximizes the score function. To that end, we define the heuristic version of the change likelihood (eq. 3) as:

$$P_h(c|\mathcal{O}) = \begin{cases} P(c|o_g), & \exists! o_g \in Gr(\mathcal{O}) \,|P(c|o_g) > 0 \\ 1 - \delta, & \nexists\, o_g \in Gr(\mathcal{O}) \,|cov(o_g, s, a) \\ 0, & \text{otherwise } (|Gr(\mathcal{O})| > 1), \end{cases} \tag{7}$$

where δ is a parameter that can trade quality for efficiency. This heuristic modifies the change likelihood (eq. 3) when no operator covers the change, giving a likelihood of $1 - \delta$ instead of 0. The heuristic score function $s_h(\mathcal{O}, T)$ is defined as the score function (eq. 6) but replacing the standard change likelihood (eq. 3) with this heuristic likelihood.

This heuristic gets the expected likelihood that can be obtained by adding new operators to $\mathcal{O}$. When $\delta = 0$, it works as an admissible heuristic (prop. 1) as it gives the maximum likelihood $= 1$ to uncovered changes. When $\delta > 0$

but close to 0, then the heuristic penalizes very specific operators when more general operators with a high likelihood are also available. The practical result is that the algorithm usually runs faster, but the heuristic is not admissible anymore.

Algorithm 1 selects the best subset of operators to explain the input transitions. In line 1, the candidate list Γ is initialized by creating one separate subset for each operator in the input set of candidates $\mathcal{O}_{input}$. Note that Γ is a set of sets of planning operators, which is initialized to $\Gamma = \{\{o_1\}, \ldots, \{o_n\}\}$ assuming that $\mathcal{O}_{input} = \{o_1, \ldots, o_n\}$. Afterwards, lines 2-3 find the best subset in Γ (which is the best set with only one operator). From that point, the candidate sets in Γ are iteratively joined together to find the best set with any number of operators, until none has $s_h(\mathcal{O}, T) > max_{score}$ (lines 4-15). In lines 5-6, the candidate $\mathcal{O}$ with the largest heuristic score is selected and removed from Γ. Then, in lines 7-8, new candidates are generated by combining the selected subset $\mathcal{O}$ with every subset in Γ. The *IsNew* method checks that the new candidate has not been already analyzed. If any of the new candidates has a new best score, it is saved as the best candidate (lines 9-11). Finally, the new candidates are added to Γ.

This method works as a search algorithm guided by an heuristic. The nodes to be analyzed are the subsets of operators stored in Γ, where they are ordered by the heuristic score value. The search tree is expanded by joining one subset with every other subset. Finally, the algorithm continues until no subset has a heuristic score larger than the best score so that the solution has been found.

The search can be used as an *anytime* algorithm, it can be stopped at any point to get the best solution found so far. Moreover, there are two options to limit in advance the processing time of the algorithm in complex problems: set a time limit, or set a limit in the size of Γ (only maintain the κ sets with the highest score in Γ). Experimental tests showed that in some domains the heuristic leads quickly to the best set, and subsequent processing is done only to confirm that no other set is better.

Property 1 *If $\delta = 0$, then the heuristic s_h is admissible: $\forall \mathcal{O}, s_h(\mathcal{O}) \geq s(\mathcal{O}^*) \mid \mathcal{O}^* = \underset{\mathcal{O}' \supset \mathcal{O}}{\operatorname{argmax}} s(\mathcal{O}')$. Therefore the optimal set will be found.*

The two parts of the score function in eq. 6 (likelihood and regularization) can be analyzed separately. Note that subsets of operators may only increase in size, as they start with one operator and can only be combined with other subsets.

- When adding operators to a set, the likelihood only increases when transition changes that were not covered before are covered by the added operators. In the heuristic score (eq. 7) all non-covered transition changes are already set to the maximum value of 1, so adding new rules cannot improve the result over the heuristic.
- The regularization part of the score function (which is $Reg(\mathcal{O}) = -\alpha \frac{Pen(\mathcal{O})}{Conf(T,\epsilon)}$) is monotonically decreasing as $Pen(\mathcal{O})$ can only increase when adding more operators, and $Conf \in (0, 1]$, $\alpha > 0$, $Pen \geq 0$.

Algorithm 1 OperatorSelection($\mathcal{O}_{input}, T$)

```
1: Current candidates Γ ← {o}, ∀ o ∈ O_input
2: max_score = max_{O∈Γ} s(O, T)
3: O_best = argmax_{O∈Γ} s(O, T)
4: while max_{O∈Γ} s_h(O, T) > max_score do
5:     O = argmax_{O∈Γ} s_h(O, T)
6:     Remove O from Γ
7:     for O' ∈ Γ | IsNew(O ∪ O') do
8:         O_new = O ∪ O'
9:         if s(O_new, T) > max_score then
10:            max_score = s(O_new, T)
11:            O_best = O_new
12:        end if
13:        Add O_new to Γ
14:    end for
15: end while
16: Output O_best
```

Property 2 *When relaxing the admissibility criterion with $\delta > 0$, the solution found by Algorithm 1 is bounded to be no worse than $\mathcal{C} \cdot \log(1 - \delta)$ plus the optimal score, where $\mathcal{C}$ is the average number of literals with the same predicate that change in a transition.*

Operators with different head predicates are analyzed separately as their effects are independent, so $\mathcal{C}$ only depends on the average changes to one predicate literals. Also note that $\delta \in [0, 1)$, and thus, $\log(1 - \delta) \leq 0$. Let $\mathcal{O}_n$ be the optimal solution with n operators and $s(\mathcal{O}_n) = opt$, then $\forall i < n$, at least one predecessor of i operators $\mathcal{O}_i$ and $s_h(\mathcal{O}_i) \geq \mathcal{C} \cdot opt + \log(1 - \delta)$ will exist.

- The regularization term is monotonically decreasing (see explanation of property 1), so $Reg(\mathcal{O}_n) \leq Reg(\mathcal{O}_i)$.
- The maximum difference between $P(\mathcal{O}_n)$ and $P_h(\mathcal{O}_i)$ is $P(t|\mathcal{O}_n) = 1$ and $P_h(t|\mathcal{O}_i) = (1 - \delta)^x$, which is the case where $\mathcal{O}_n$ has perfect coverage, $\mathcal{O}_i$ has no coverage (all the coverage is obtained from $\mathcal{O}_n \setminus \mathcal{O}_i$) and the transition has x changes. Then, if we take the worst case for all transitions, and an average of $\mathcal{C}$ changes per transition, $P(\mathcal{O}_n) - P_h(\mathcal{O}_i) = E[\log(P(T|\mathcal{O}_n))] - E[\log(P_h(T|\mathcal{O}_i))] = \log(1) - \log(1-\delta)^{\mathcal{C}} = \mathcal{C} \cdot \log(1-\delta)$.

Therefore the predecessors of the optimal solution will be checked (and thus the optimal solution found) unless a solution with $s(\mathcal{O}) \geq opt + c \cdot log(1 - \delta)$ is found before.

A value of $\delta > 0$ can speed up the algorithm considerably at the expense of optimality. When two operators have similar likelihoods, $\delta > 0$ penalizes the most specific one. This results in general rule sets being analyzed first, and thus models with better likelihoods are obtained earlier.

Subsumption Tree

In this section we present a method to speed up the approach by partitioning the set of candidates into smaller groups. The idea of the subsumption tree is to start with the best subset of

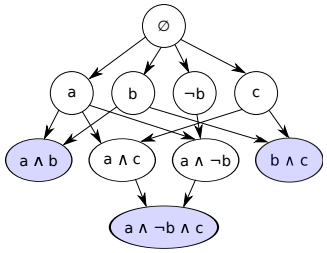

Figure 2: Example of a subsumption tree. Each letter (a, b, c) represents a literal in the extended body of a planning operator. The leaves are the nodes painted in blue.

Algorithm 2 OperatorSelectionSubsumption($Tree_{\mathcal{O}}, T$)

1: **do**
2: $\quad\mathcal{O}_L = leaves(Tree_{\mathcal{O}})$
3: $\quad\mathcal{O}_{L,best} = \texttt{OperatorSelection}(\mathcal{O}_L, T)$
4: $\quad$Remove $(\mathcal{O}_L \setminus \mathcal{O}_{L,best})$ from $Tree_{\mathcal{O}}$
5: $\quad\mathcal{O}_P = \mathcal{O}_{L,best} \cup parents(Tree_{\mathcal{O}}, \mathcal{O}_{L,best})$
6: $\quad\mathcal{O}_{P,best} = \texttt{OperatorSelection}(\mathcal{O}_P, T)$
7: $\quad$Remove $(\mathcal{O}_L \setminus \mathcal{O}_{P,best})$ from $Tree_{\mathcal{O}}$
8: **while** $Tree_{\mathcal{O}}$ changed
9: **Output** $= leaves(Tree_{\mathcal{O}})$

specialized operators, and iteratively check if more general operators yield better scores.

Definition 1 (Subsumption relation) *First, we define the extended body of an operator as* $body_{ext}(o) = body(o)(\wedge a)$ *where* $(\wedge a)$ *only appears if o has an action. Let* o_1 *and* o_2 *be two planning operators with* $head(o_1) = head(o_2)$*,* o_1 *is subsumed by* o_2 *if* $(body_{ext}(o_1) \supseteq body_{ext}(o_2))$*.*

Definition 2 (Subsumption tree) *The subsumption tree* $Tree_{\mathcal{O}}$ *of a set of planning operators* $\mathcal{O} = \{o_1, ..., o_n\}$ *is a directed graph with arcs* (o_i, o_j) *when* o_i *subsumes* o_j *and* $|body_{ext}(o_j)| - |body_{ext}(o_i)| = 1$*. We call the set of leaves* $L(Tree_{\mathcal{O}})$*. Figure 2 shows an example of a subsumption tree.*

The subsumption tree orders the rules in levels that represent the generality of the operators: the less literals the more general the operator. Based on this tree, Algorithm 2 selects the operators. The idea is to start by identifying the best specific operators, and then check if their generalizations improve the results. In lines 2-4, the best subset of leaves $\mathcal{O}_{L,best}$ is identified, and all leaves not in $\mathcal{O}_{L,best}$ are removed from the tree. Then, in lines 5-7, a new set of operators $\mathcal{O}_P$ is created that includes $\mathcal{O}_{L,best}$ and the operators that subsume them (their parents in the tree). The best subset $\mathcal{O}_{P,best}$ in $\mathcal{O}_P$ is selected, and $\mathcal{O}_L \setminus \mathcal{O}_{P,best}$ are removed. This is repeated until nothing is changed in the tree.

The performance is improved by using the subsumption tree: it divides the candidates into subsets, and the planning operator selection is much faster with smaller sets of candidates. Although it sacrifices optimality, experimental tests showed that the results obtained were in many cases optimal or near optimal. This happens due to the fact that in most cases $P(\mathcal{O}) >> Reg(\mathcal{O})$. The operators in the leaves maximize $P(\mathcal{O})$ as they are more specialized, while the operators near the root maximize $Reg(\mathcal{O})$ as they are more general. Thus, the subset of leaves selected in the first iteration usually is near optimal, and afterwards the method only has to find the right level of generalization.

Note that if the subsumption tree is used, the best operators may be close to the root of the tree (and thus they will be analyzed at the end), so the learner shouldn't be used as an anytime algorithm. However, the processing time can still be bounded with satisfactory results by limiting the size of Γ to κ sets in Algorithm 1.

5 Experiments

This section describes the experimental evaluation of our approach. The learner was applied to three domains of the 2014 International Probabilistic Planning Competition (IPPC).

The evaluation follows the same scheme of (Pasula, Zettlemoyer, and Kaelbling 2007). To learn the domains, a set of state transitions $(s, a, s') \in T$ is generated randomly. To create a transition, first the state s is constructed by randomly assigning a value (positive or negative) to every literal, but ensuring that the resulting state is valid (e.g. in the elevators domain, an elevator cannot be in two different floors at the same time). Then, the action a arguments are picked randomly, and the state s' is obtained by applying all operators to (s, a). The distribution used to construct s is biased to guarantee that, in at least half of the examples, the operators that contain a have a chance of changing the state.

The evaluation of the learned models is carried out by calculating the *average variational distance* between the true model $\mathcal{O}$ and the estimate $\hat{\mathcal{O}}$. This evaluation uses a new set of similarly generated random transitions T':

$$D(\mathcal{O}, \hat{\mathcal{O}}) = \frac{1}{|T'|} \sum_{t \in T'} \left| P(t|\mathcal{O}) - P(t|\hat{\mathcal{O}}) \right|. \qquad (8)$$

Domains

Three IPPC 2014 domains were used in the experiments. Note that they were slightly modified to remove redundancy (e.g. a *north(?X,?Y)* literal is equivalent to *south(?Y,?X)*, so one can be replaced by the other).

- *Triangle Tireworld*. This domain is the easiest one, it has uncertain effects, but no exogenous effects. It is modeled with 5 different predicates, 3 actions, 7 operators, and operators require at most 2 terms ($\omega = 2$). This domain serves as a good baseline to compare with the state of the art as there are not exogenous effects.
- *Crossing Traffic*. This domain has an intermediate difficulty. It has uncertain effects and exogenous effects, which makes it more challenging, but the complexity of the model is still moderate: 8 predicates, 4 actions, 6 operators, and operators take at most 3 terms.
- *Elevators*. This is the most challenging domain. It has uncertain effects and exogenous effects. It is modeled with 10 predicates, 4 actions, 17 operators, and operators take at most 3 terms.

The difficulty to learn a domain is mostly given by:

- The maximum number of terms ω that operators may have. The number of terms increases exponentially the number of relational transitions generated from the input grounded transitions (Sec. 3), and therefore the number of candidate rules. If the value of ω is larger than the number of terms that operators actually require, the learning time increases while the quality of the models remains the same.
- The number of predicates, both constant and variable, used to represent the states. The candidates that LFIT generates consider all combinations of predicates that are consistent with the transitions.
- The number of uncertain and exogenous effects. LFIT generates all candidates that may explain an effect, including operators that overfit and underfit, and all combinations of action and exogenous effects.

Evaluation

In this section we analyze experimentally the proposed algorithm, its parameters, and how it compares with the state of the art. The learner uses the following parameters: α, ϵ, and δ, which are the score function parameters; κ, which is the size limit of Γ (Algorithm 1); and "$tree$" to denote that the subsumption tree is being used.

The results show the *average variational distance*, which may be difficult to interpret. To give an idea about the utility of the learned models, a planner can usually yield a plan when the *average variational distance* is below:

- 0.09 in the *Triangle Tireworld* domain.
- 0.15 in the *Crossing Traffic* domain.
- 0.1 in the *Elevators* domain.

As the *average variational distance* becomes lower, there is a higher probability that obtained plans are optimal.

Configuration parameters. Here we discuss the impact the different configuration parameters have on the quality of the models learned.

- As seen in figs. 3-left and 4-left, the confidence term in the score function improves the quality of the models learned when few input transitions are available. This confidence term penalizes very specialized rules in cases where there is a large uncertainty in the predictions. Once many transitions are available, the impact of this term disappears. Note that only probabilistic operators are improved as deterministic ones are completely specialized anyway. The impact of this term is relatively small in the experiments because both domains require only one probabilistic operator.
- Figure 3-middle shows how the optimal search configuration compares to one that uses the subsumption tree, a non-admissible heuristic ($\delta > 0$), and restricts the size of Γ to κ sets. Both yield almost the same results because the subsumption tree and the relaxed heuristic degrade the results only in very specific cases, and the value of κ is high enough to find the relevant operators.

  The Triangle Tireworld domain does not contain exogenous events, and thus, the selection of the best set of candidates is very simple and can be done equally fast in the optimal case (fig. 3-right). Most of the execution time is spent in obtaining the likelihood of the operators as there are a lot of possible candidates in this domain.
- In contrast, when learning domains with exogenous events, the optimal search method was not used because the execution time was too large in all but the most simple problems. Similar results to the optimal method were obtained when the set of possible candidates Γ was restricted to a size κ that was large enough, and the execution time was greatly reduced.

  Figure 4-middle shows the advantages of the subsumption tree and a non-admissible heuristic ($\delta > 0$). The subsumption tree divides the search problem in smaller ones, and thus a lower κ is enough to yield good results. The same applies to the non-admissible heuristic since it prioritizes operators that explain many transitions with a high likelihood, and very specific operators that may not be interesting are given a smaller heuristic score. As a result, the configurations without the subsumption tree and $\delta = 0$ are slower because they analyze more operators to obtain the same results (fig. 4-right).

  As an exceptional case, LFIT takes longer to execute in the Crossing Traffic domain with very few transitions as it finds more possible patterns. Its execution time increases again once many transitions have been added.

Comparison with Pasula et al. Comparison is performed only with (Pasula, Zettlemoyer, and Kaelbling 2007), as (Deshpande et al. 2007) is an extension to include transfer learning, and (Mourao 2014) yields similar results to Pasula et al.'s approach in completely observable problems. The experiments were done with the implementation by Lang and Toussaint (2010).

Figure 5-left shows the results of learning the Triangle Tireworld domain. It can be easily learned by both, ours, and Pasula et al.'s approach, as there are no exogenous effects. With this experiment we want to show that our method can learn domains without exogenous effects as well as state-of-the-art learners.

Figures 5-center and 5-right refer to two domains with exogenous effects. As Pasula et al.'s learner cannot learn exogenous effects, it tries to build overcomplicated operators that explain both action effects and exogenous effects at the same time, and thus it is not able to yield a good general model. In contrast, our approach is able to distinguish action effects from exogenous effects once enough transitions are given as input.

6 Conclusions

We have introduced a new method that, given a set of input transitions, learns a general model explaining them. In contrast to previous approaches, it can learn exogenous effects (effects not related to any action), while still being similarly good at obtaining a relational representation of the problem and at learning uncertain effects. Moreover optimal and suboptimal search methods are provided, so the best approach

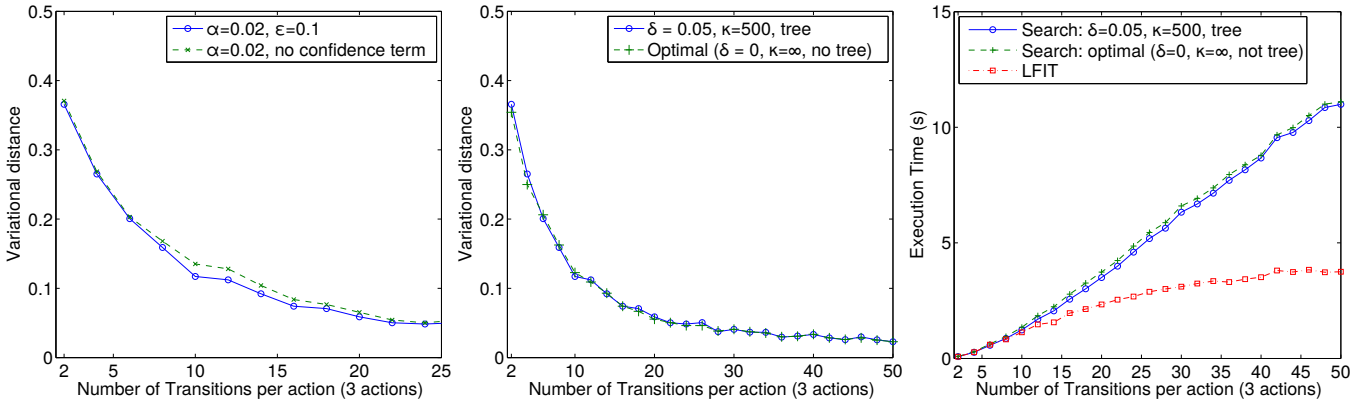

Figure 3: Evaluation of different configurations in the Triangle Tireworld domain, MDP-1. Unless stated otherwise, the following parameters were used: $(\alpha = 0.02,\ \epsilon = 0.1,\ \omega = 2,\ \delta = 0.05,\ \kappa = 500, \text{tree})$. The results shown are the means obtained from 100 runs. The evaluation was done with 3000 transitions. **Left:** Influence of the confidence term. **Center:** Comparing optimal and suboptimal search configurations. **Right:** Execution time of the search (optimization) and the LFIT modules. The total learning time would be the sum of LFIT plus one of the search configurations.

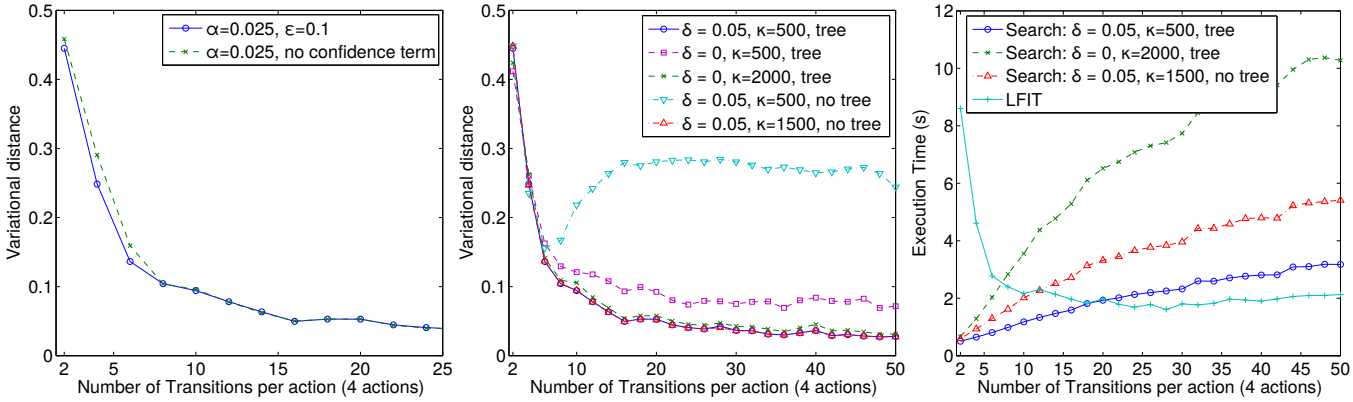

Figure 4: Evaluation of different configurations in the Crossing Traffic domain, MDP-1. Unless stated otherwise, the following parameters were used: $(\alpha = 0.025,\ \epsilon = 0.1,\ \omega = 3,\ \delta = 0.05,\ \kappa = 500, \text{tree})$. The results shown are the means obtained from 100 runs. The evaluation was done with 4000 transitions. **Left:** Influence of the confidence term. **Center:** Influence of δ and the subsumption tree. **Right:** Execution time of the search (optimization) and the LFIT modules. The total learning time would be the sum of LFIT plus one of the search configurations.

can be chosen depending on the quality requirements, the difficulty of the problem, and the learning time available.

The main limitation of the algorithm is scalability. If the number of generated propositional predicates is large (either because the domain is represented with a large number of predicates, or because ω takes a high value), the problem may become intractable. Moreover, as the universal quantifier (*forall*) and the negative existential quantifier ($\sim$ *exists*) are not supported, many IPPC 2014 domains cannot be learned.

Experimental validation was carried out with three standard domains of the planning community. Our approach could learn domains with exogenous effects where previous approaches could not. The experiments also show the improvements obtained with the subsumption tree and the proposed confidence term in the score function.

As future work, the following topics are proposed: improving the transformation from propositional rules to planning operators to learn universal and existential quantifiers, use a multi-valued relational representation (currently only the propositional part is multi-valued), and support partial observability.

Acknowledgements

This work has been supported by the MINECO project RobInstruct TIN2014-58178-R and the ERA-Net CHIST-ERA project I-DRESS PCIN-2015-147. D. Martínez is also supported by the Spanish Ministry of Education, Culture and Sport via a FPU doctoral grant (FPU12-04173).

References

Deshpande, A.; Milch, B.; Zettlemoyer, L. S.; and Kaelbling, L. P. 2007. Learning probabilistic relational dynamics

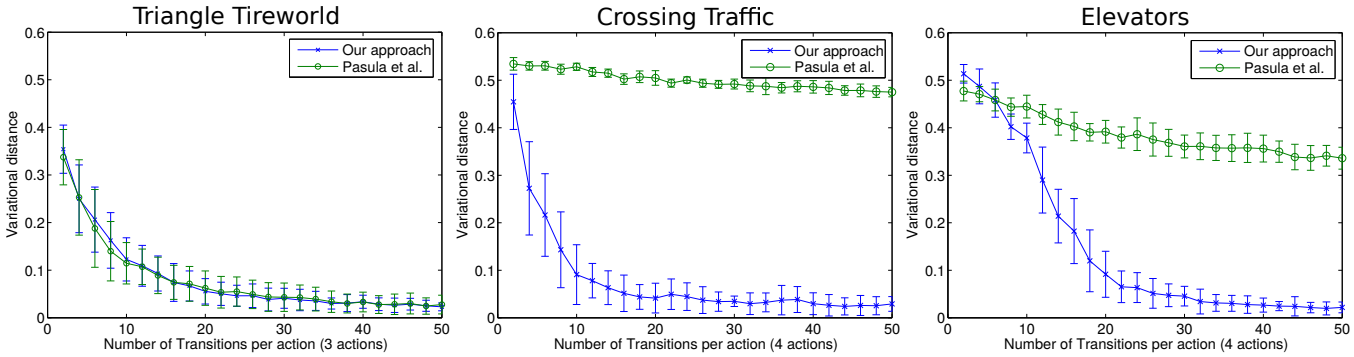

Figure 5: Comparison with Pasula et al. The results shown are the means and standard deviations obtained from 50 runs. The evaluation was done with 5000 random transitions. **Left:** Triangle Tireworld ($\alpha = 0.02$, $\epsilon = 0.1$, $\omega = 2$, $\delta = 0$, $\kappa = \infty$, no tree). **Center:** Crossing Traffic ($\alpha = 0.025$, $\epsilon = 0.1$, $\omega = 3$, $\delta = 0.05$, $\kappa = 500$, tree). **Right:** Elevators ($\alpha = 0.015$, $\epsilon = 0.1$, $\omega = 3$, $\delta = 0.05$, $\kappa = 500$, tree).

for multiple tasks. In *Proc. of the Conference on Uncertainty in Artificial Intelligence*.

Inoue, K.; Ribeiro, T.; and Sakama, C. 2014. Learning from interpretation transition. *Machine Learning* 94(1):51–79.

Keller, T., and Eyerich, P. 2012. PROST: Probabilistic Planning Based on UCT. In *Proc. of the International Conference on Automated Planning and Scheduling*, 119–127.

Kolobov, A.; Dai, P.; Mausam; and Weld, D. S. 2012. Reverse iterative deepening for finite-horizon MDPs with large branching factors. In *Proc. of the International Conference on Automated Planning and Scheduling*, 146–154.

Kulick, J.; Toussaint, M.; Lang, T.; and Lopes, M. 2013. Active learning for teaching a robot grounded relational symbols. In *Proc. of the Twenty-Third international joint conference on Artificial Intelligence*, 1451–1457.

Lang, T., and Toussaint, M. 2010. Planning with noisy probabilistic relational rules. *Journal of Artificial Intelligence Research* 39:1–49.

Martínez, D.; Alenyà, G.; and Torras, C. 2015. Planning robot manipulation to clean planar surfaces. *Engineering Applications of Artificial Intelligence* 39:23–32.

Martínez, D.; Ribeiro, T.; Inoue, K.; Alenyà, G.; and Torras, C. 2015. Learning probabilistic action models from interpretation transitions. In *Technical Communication of the International Conference on Logic Programming, CEUR Workshop Proceedings*, volume 1433, 30.

Mourao, K.; Zettlemoyer, L. S.; Petrick, R.; and Steedman, M. 2012. Learning strips operators from noisy and incomplete observations. In *Proc. of the Conference on Uncertainty in Artificial Intelligence*, 614–623.

Mourao, K. 2014. Learning probabilistic planning operators from noisy observations. In *Proc. of the Workshop of the UK Planning and Scheduling Special Interest Group*.

Pasula, H. M.; Zettlemoyer, L. S.; and Kaelbling, L. P. 2007. Learning symbolic models of stochastic domains. *Journal of Artificial Intelligence Research* 29(1):309–352.

Ribeiro, T., and Inoue, K. 2014. Learning prime implicant conditions from interpretation transition. In *Proc. of the International Conference on Inductive Logic Programming*.

Sanner, S. 2010. Relational dynamic influence diagram language (RDDL): Language description. *Unpublished ms. Australian National University*.

Sykes, D.; Corapi, D.; Magee, J.; Kramer, J.; Russo, A.; and Inoue, K. 2013. Learning revised models for planning in adaptive systems. In *Proc. of the International Conference on Software Engineering*, 63–71.

Thon, I.; Gutmann, B.; van Otterlo, M.; Landwehr, N.; and De Raedt, L. 2009. From non-deterministic to probabilistic planning with the help of statistical relational learning. In *Proc. of the ICAPS Workshop on Planning and Learning*, 23–30.

Thon, I.; Landwehr, N.; and De Raedt, L. 2011. Stochastic relational processes: Efficient inference and applications. *Machine Learning* 82(2):239–272.

Walsh, T. J.; Szita, I.; Diuk, C.; and Littman, M. L. 2009. Exploring compact reinforcement-learning representations with linear regression. In *Proc. of the Conference on Uncertainty in Artificial Intelligence*, 591–598.

Walsh, T. J. 2010. *Efficient learning of relational models for sequential decision making*. Ph.D. Dissertation, Rutgers, The State University of New Jersey.

Younes, H. L., and Littman, M. L. 2004. PPDDL1. 0: An extension to PDDL for expressing planning domains with probabilistic effects. *Technical Report CMU-CS-04-162*.

Zhu, G.; Lizotte, D.; and Hoey, J. 2014. Scalable approximate policies for Markov decision process models of hospital elective admissions. *Artificial Intelligence in Medicine* 61(1):21–34.

Zhuo, H. H., and Kambhampati, S. 2013. Action-model acquisition from noisy plan traces. In *Proc. of the Twenty-Third international joint conference on Artificial Intelligence*, 2444–2450.

Zhuo, H. H., and Yang, Q. 2014. Action-model acquisition for planning via transfer learning. *Artificial intelligence* 212:80–103.